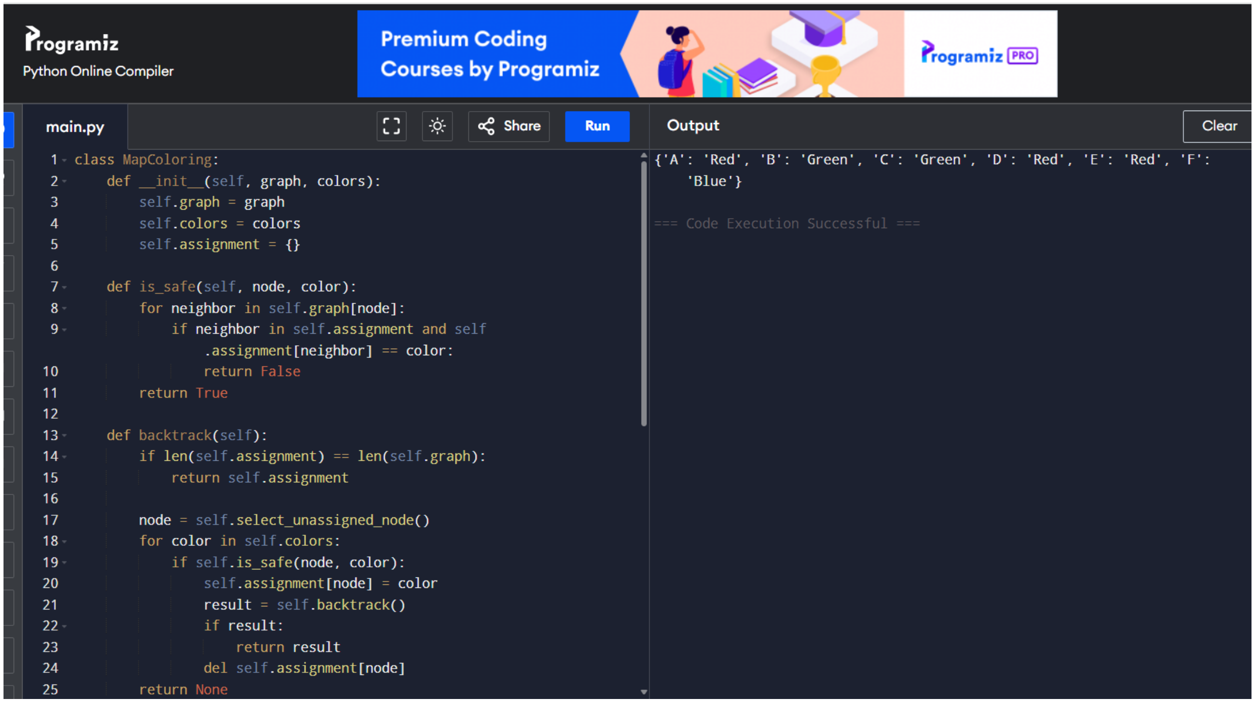
11. Write the python program for map coloring to implement CSP.

```
class MapColoring:
    def __init__(self, graph, colors):
        self.graph = graph
        self.colors = colors
        self.assignment = {}

    def is_safe(self, node, color):
        for neighbor in self.graph[node]:
            if neighbor in self.assignment and self.assignment[neighbor] == color:
                return False
        return True

    def backtrack(self):
        if len(self.assignment) == len(self.graph):
            return self.assignment

        node = self.select_unassigned_node()
        for color in self.colors:
            if self.is_safe(node, color):
                self.assignment[node] = color
                result = self.backtrack()
                if result:
                    return result
                del self.assignment[node]
        return None
    def select_unassigned_node(self):
        for node in self.graph:
            if node not in self.assignment:
                return node
        return None

    def solve(self):
        return self.backtrack()

# Example usage
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}
colors = ['Red', 'Green', 'Blue']

map_coloring = MapColoring(graph, colors)
solution = map_coloring.solve()
print(solution)
```

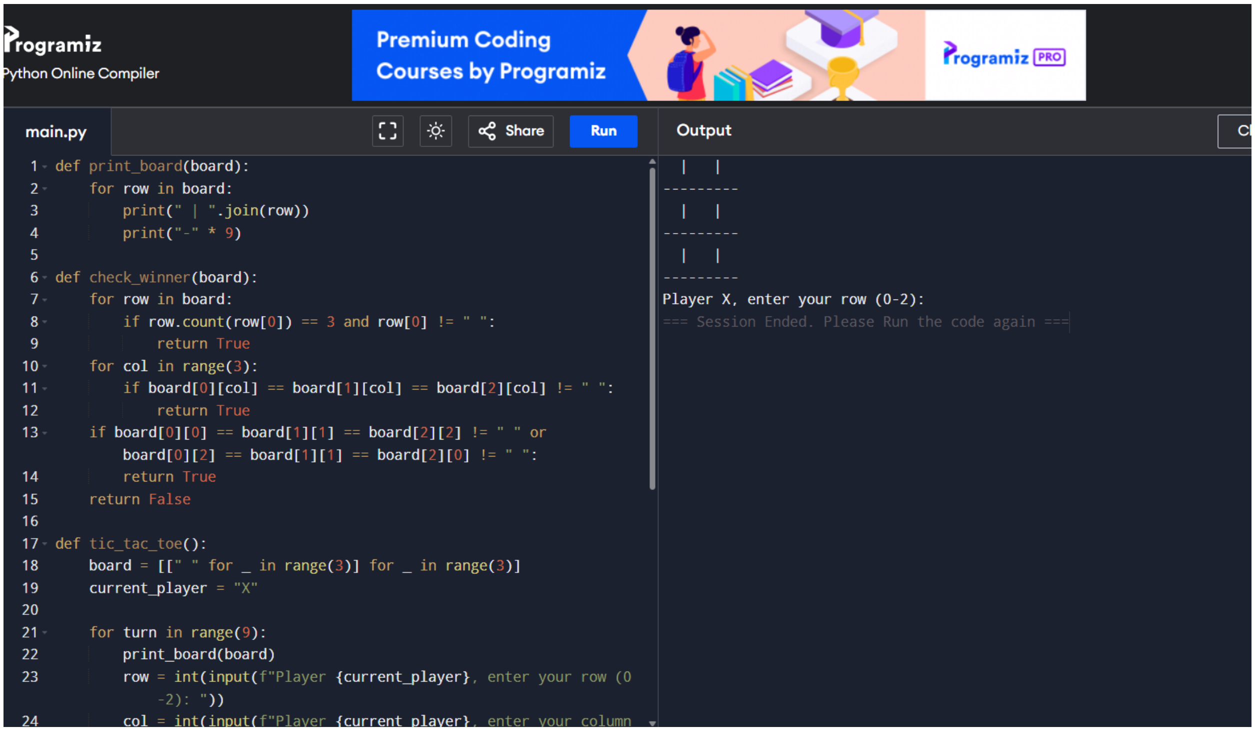


```python
class MapColoring:
    def __init__(self, graph, colors):
        self.graph = graph
        self.colors = colors
        self.assignment = {}

    def is_safe(self, node, color):
        for neighbor in self.graph[node]:
            if neighbor in self.assignment and self
                .assignment[neighbor] == color:
                return False
        return True

    def backtrack(self):
        if len(self.assignment) == len(self.graph):
            return self.assignment

        node = self.select_unassigned_node()
        for color in self.colors:
            if self.is_safe(node, color):
                self.assignment[node] = color
                result = self.backtrack()
                if result:
                    return result
                del self.assignment[node]
        return None
```

```
{'A': 'Red', 'B': 'Green', 'C': 'Green', 'D': 'Red', 'E': 'Red', 'F':
    'Blue'}

=== Code Execution Successful ===
```

12. Write the python program for Tic Tac Toc game.

```python
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)


def check_winner(board):
    for row in board:
        if row.count(row[0]) == 3 and row[0] != " ":
            return True
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != " ":
            return True
    if board[0][0] == board[1][1] == board[2][2] != " " or board[0][2] == board[1][1] == board[2][0] != " ":
        return True
    return False

def tic_tac_toe():
    board = [[" " for _ in range(3)] for _ in range(3)]
    current_player = "X"

    for turn in range(9):
        print_board(board)
        row = int(input(f"Player {current_player}, enter your row (0-2): "))
        col = int(input(f"Player {current_player}, enter your column (0-2): "))

        if board[row][col] == " ":
            board[row][col] = current_player
            if check_winner(board):
                print_board(board)
                print(f"Player {current_player} wins!")
                return
            current_player = "O" if current_player == "X" else "X"
        else:
            print("Invalid move, try again.")

    print_board(board)
    print("It's a draw!")

if __name__ == "__main__":
 tic_tac_toe()
```



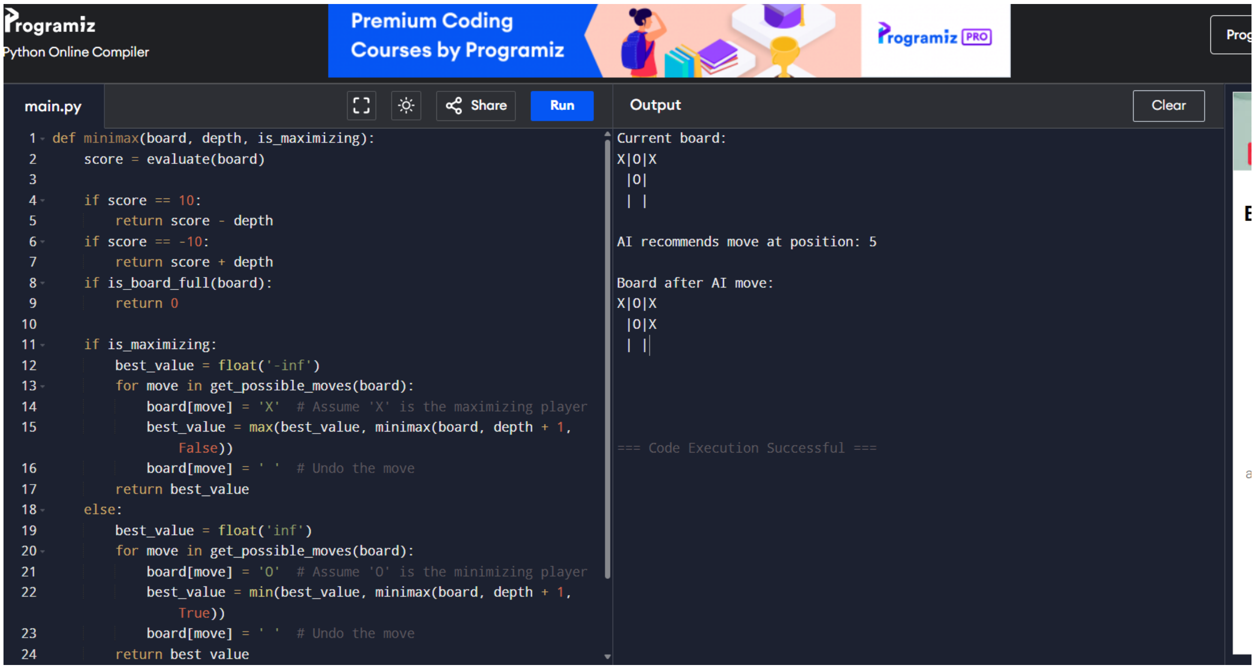
```python
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)

def check_winner(board):
    for row in board:
        if row.count(row[0]) == 3 and row[0] != " ":
            return True
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != " ":
            return True
    if board[0][0] == board[1][1] == board[2][2] != " " or
        board[0][2] == board[1][1] == board[2][0] != " ":
        return True
    return False

def tic_tac_toe():
    board = [[" " for _ in range(3)] for _ in range(3)]
    current_player = "X"

    for turn in range(9):
        print_board(board)
        row = int(input(f"Player {current_player}, enter your row (0-2): "))
        col = int(input(f"Player {current_player}, enter your column
```

```
  |   |
---------
  |   |
---------
  |   |
---------
Player X, enter your row (0-2):
=== Session Ended. Please Run the code again ===
```

13. write the python program to implement minimax algorithm for gaming.

```
def minimax(board, depth, is_maximizing):
    score = evaluate(board)


    if score == 10:
        return score - depth
    if score == -10:
        return score + depth
    if is_board_full(board):
        return 0


    if is_maximizing:
        best_value = float('-inf')
        for move in get_possible_moves(board):
            board[move] = 'X'  # Assume 'X' is the maximizing player
            best_value = max(best_value, minimax(board, depth + 1, False))
            board[move] = ' '  # Undo the move
        return best_value
    else:
        best_value = float('inf')
        for move in get_possible_moves(board):
            board[move] = 'O'  # Assume 'O' is the minimizing player
            best_value = min(best_value, minimax(board, depth + 1, True))
            board[move] = ' '  # Undo the move
        return best_value
```

# Additional functions like evaluate, is_board_full, and get_possible_moves need to be defined.

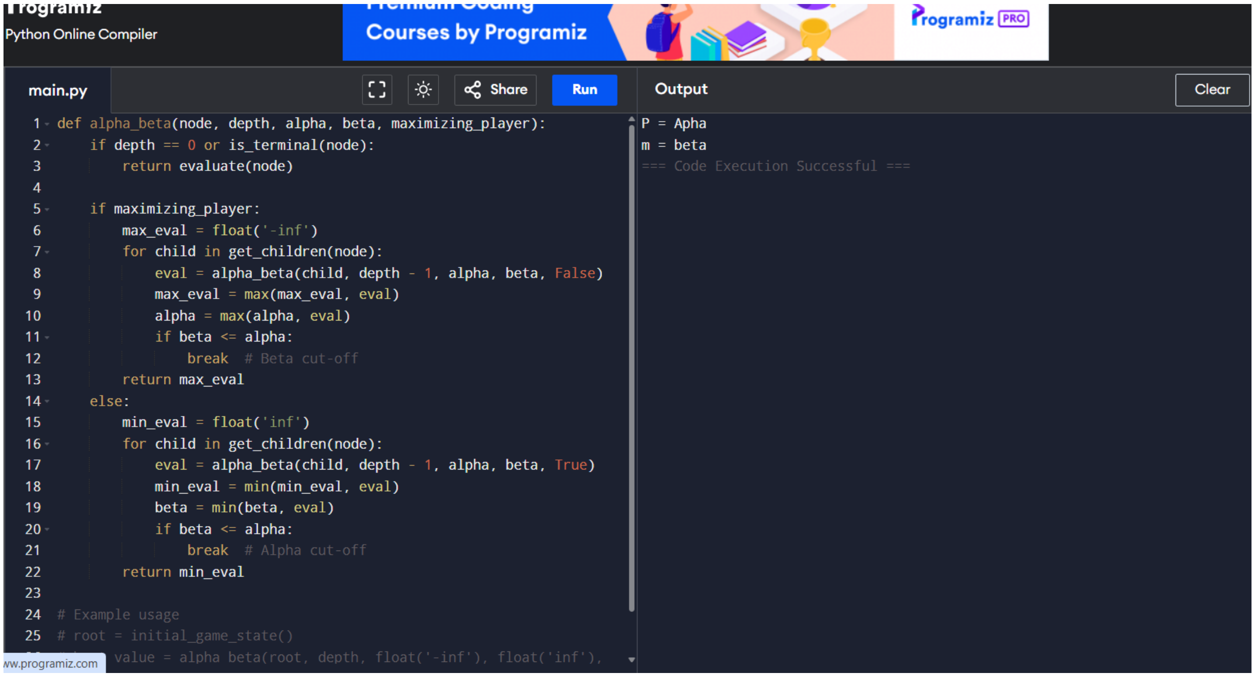


```
def minimax(board, depth, is_maximizing):
    score = evaluate(board)

    if score == 10:
        return score - depth
    if score == -10:
        return score + depth
    if is_board_full(board):
        return 0

    if is_maximizing:
        best_value = float('-inf')
        for move in get_possible_moves(board):
            board[move] = 'X'  # Assume 'X' is the maximizing player
            best_value = max(best_value, minimax(board, depth + 1,
                False))
            board[move] = ' '  # Undo the move
        return best_value
    else:
        best_value = float('inf')
        for move in get_possible_moves(board):
            board[move] = 'O'  # Assume 'O' is the minimizing player
            best_value = min(best_value, minimax(board, depth + 1,
                True))
            board[move] = ' '  # Undo the move
        return best_value
```

```
Current board:
X|O|X
 |O|
 | |

AI recommends move at position: 5

Board after AI move:
X|O|X
 |O|X
 | |


=== Code Execution Successful ===
```

14. Write the python program to implement Apha and beta pruning algorithm for gaming .

```
def alpha_beta(node, depth, alpha, beta, maximizing_player):
    if depth == 0 or is_terminal(node):
        return evaluate(node)

    if maximizing_player:
        max_eval = float('-inf')
        for child in get_children(node):
            eval = alpha_beta(child, depth - 1, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break  # Beta cut-off
        return max_eval
    else:
        min_eval = float('inf')
        for child in get_children(node):
            eval = alpha_beta(child, depth - 1, alpha, beta, True)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break  # Alpha cut-off
        return min_eval

# Example usage
# root = initial_game_state()
# best_value = alpha_beta(root, depth, float('-inf'), float('inf'), True)
```



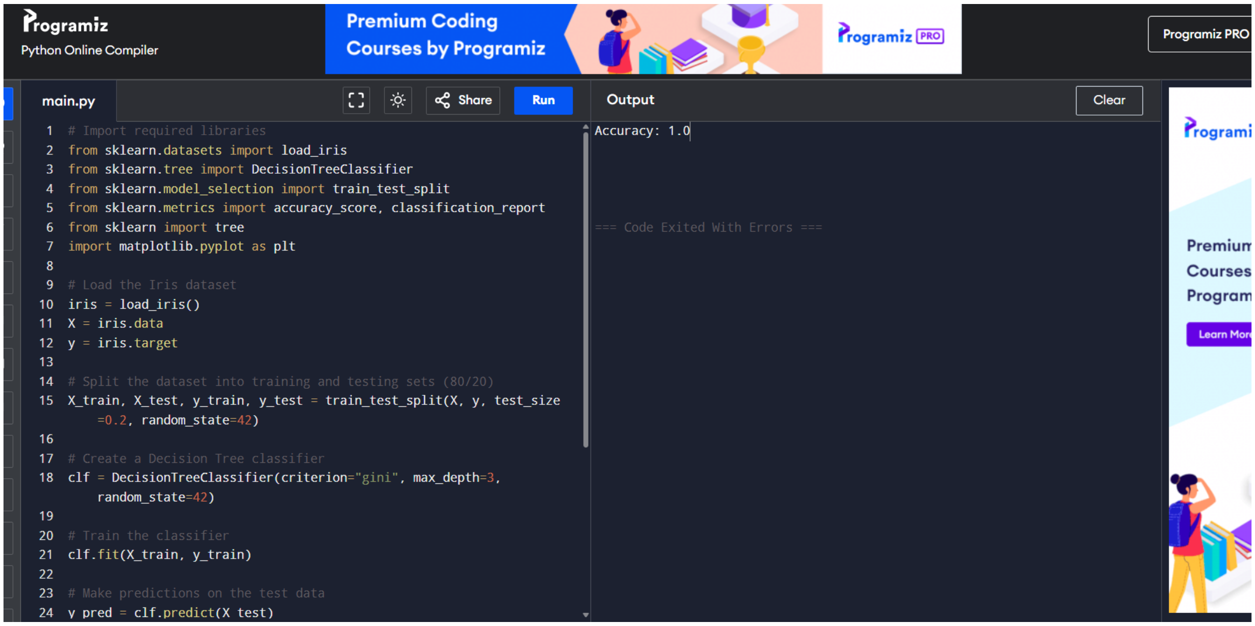
```
def alpha_beta(node, depth, alpha, beta, maximizing_player):
    if depth == 0 or is_terminal(node):
        return evaluate(node)

    if maximizing_player:
        max_eval = float('-inf')
        for child in get_children(node):
            eval = alpha_beta(child, depth - 1, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break  # Beta cut-off
        return max_eval
    else:
        min_eval = float('inf')
        for child in get_children(node):
            eval = alpha_beta(child, depth - 1, alpha, beta, True)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break  # Alpha cut-off
        return min_eval

# Example usage
# root = initial_game_state()
  value = alpha beta(root, depth, float('-inf'), float('inf'),
```

```
P = Apha
m = beta
=== Code Execution Successful ===
```

15. Write the python program to implement decision tree.

```
# Import necessary libraries

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

import matplotlib.pyplot as plt


# Load the Iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Create a Decision Tree Classifier

clf = DecisionTreeClassifier()


# Train the model

clf.fit(X_train, y_train)


# Evaluate the model

accuracy = clf.score(X_test, y_test)

print(f"Accuracy: {accuracy:.2f}")


# Visualize the decision tree

plt.figure(figsize=(12,8))

tree.plot_tree(clf, filled=True)

plt.show()
```

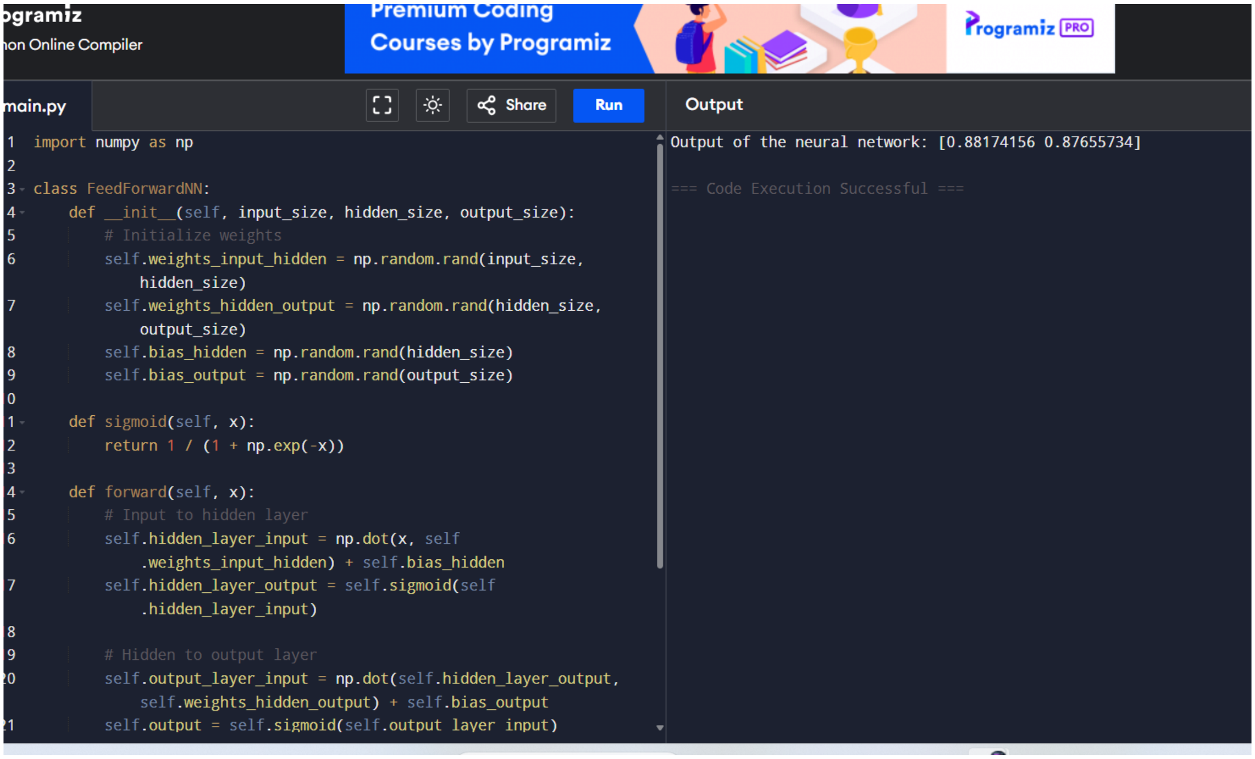


```
# Import required libraries
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn import tree
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)

# Create a Decision Tree classifier
clf = DecisionTreeClassifier(criterion="gini", max_depth=3,
    random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)
```

```
Accuracy: 1.0



=== Code Exited With Errors ===
```

16. Write the python program to implement feed forward neural network.

```
import numpy as np


class FeedForwardNN:
    def __init__(self, input_size, hidden_size, output_size):
        # Initialize weights
        self.weights_input_hidden = np.random.rand(input_size, hidden_size)
        self.weights_hidden_output = np.random.rand(hidden_size, output_size)
        self.bias_hidden = np.random.rand(hidden_size)
        self.bias_output = np.random.rand(output_size)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def forward(self, x):
        # Input to hidden layer
        self.hidden_layer_input = np.dot(x, self.weights_input_hidden) + self.bias_hidden
        self.hidden_layer_output = self.sigmoid(self.hidden_layer_input)

        # Hidden to output layer
        self.output_layer_input = np.dot(self.hidden_layer_output, self.weights_hidden_output) +
self.bias_output
        self.output = self.sigmoid(self.output_layer_input)
        return self.output

# Example usage
if __name__ == "__main__":
    nn = FeedForwardNN(input_size=3, hidden_size=5, output_size=2)

    input_data = np.array([0.1, 0.2, 0.3])

    output = nn.forward(input_data)

    print("Output of the neural network:", output)
```

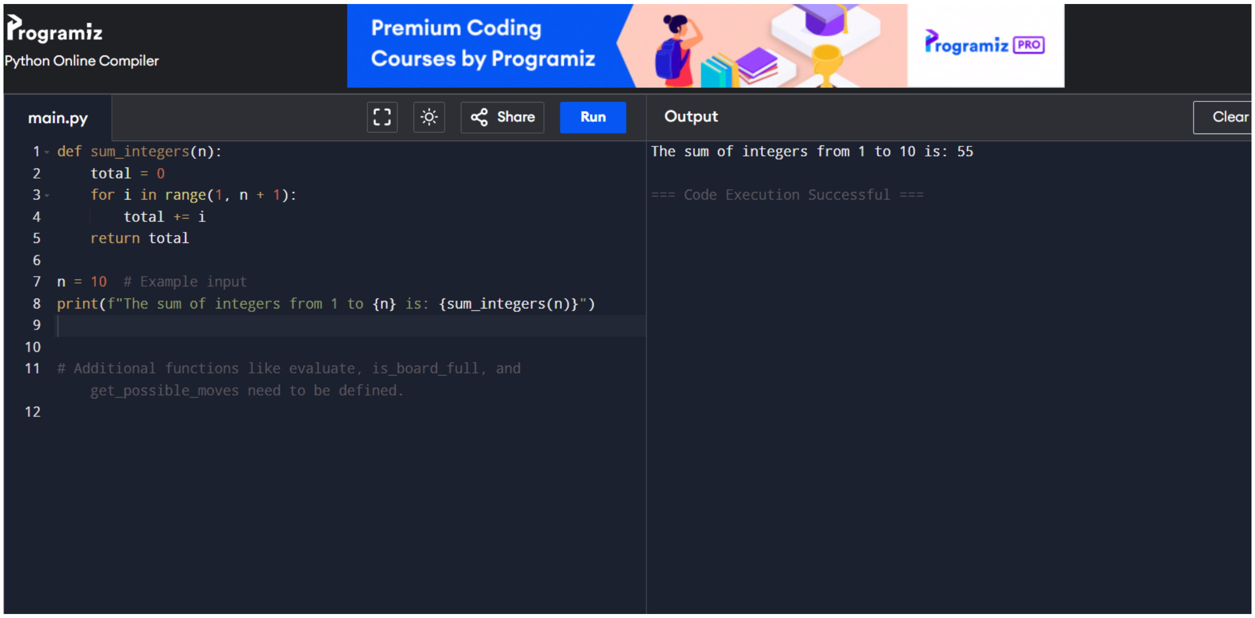


```
import numpy as np

class FeedForwardNN:
    def __init__(self, input_size, hidden_size, output_size):
        # Initialize weights
        self.weights_input_hidden = np.random.rand(input_size,
            hidden_size)
        self.weights_hidden_output = np.random.rand(hidden_size,
            output_size)
        self.bias_hidden = np.random.rand(hidden_size)
        self.bias_output = np.random.rand(output_size)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def forward(self, x):
        # Input to hidden layer
        self.hidden_layer_input = np.dot(x, self
            .weights_input_hidden) + self.bias_hidden
        self.hidden_layer_output = self.sigmoid(self
            .hidden_layer_input)

        # Hidden to output layer
        self.output_layer_input = np.dot(self.hidden_layer_output,
            self.weights_hidden_output) + self.bias_output
        self.output = self.sigmoid(self.output_layer_input)
```

```
Output of the neural network: [0.88174156 0.87655734]

=== Code Execution Successful ===
```

17. Write the python program to sum the integers from 1 to n.

```
def sum_integers(n):

    total = 0

    for i in range(1, n + 1):

        total += i

    return total


n = 10  # Example input

print(f"The sum of integers from 1 to {n} is: {sum_integers(n)}")
```

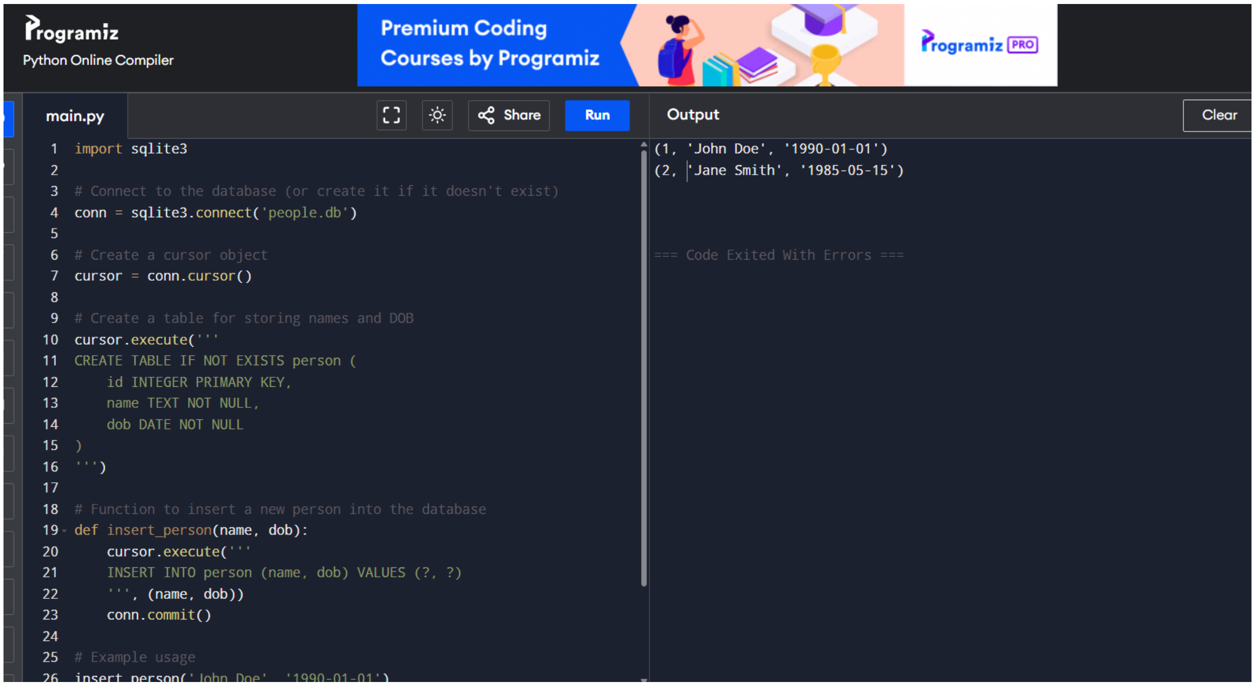


```python
def sum_integers(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

n = 10  # Example input
print(f"The sum of integers from 1 to {n} is: {sum_integers(n)}")


# Additional functions like evaluate, is_board_full, and get_possible_moves need to be defined.

```

```
The sum of integers from 1 to 10 is: 55

=== Code Execution Successful ===
```

18. Write the prolog program for A DB WITH NAME,DOB.

```
import sqlite3


# Connect to the database (or create it if it doesn't exist)

conn = sqlite3.connect('people.db')


# Create a cursor object

cursor = conn.cursor()


# Create a table for storing names and DOB

cursor.execute('''

CREATE TABLE IF NOT EXISTS person (

    id INTEGER PRIMARY KEY,

    name TEXT NOT NULL,

    dob DATE NOT NULL

)
''')

# Function to insert a new person into the database
def insert_person(name, dob):
    cursor.execute('''
    INSERT INTO person (name, dob) VALUES (?, ?)
    ''', (name, dob))
    conn.commit()

# Example usage
insert_person('John Doe', '1990-01-01')
insert_person('Jane Smith', '1985-05-15')

# Close the connection
conn.close()
```



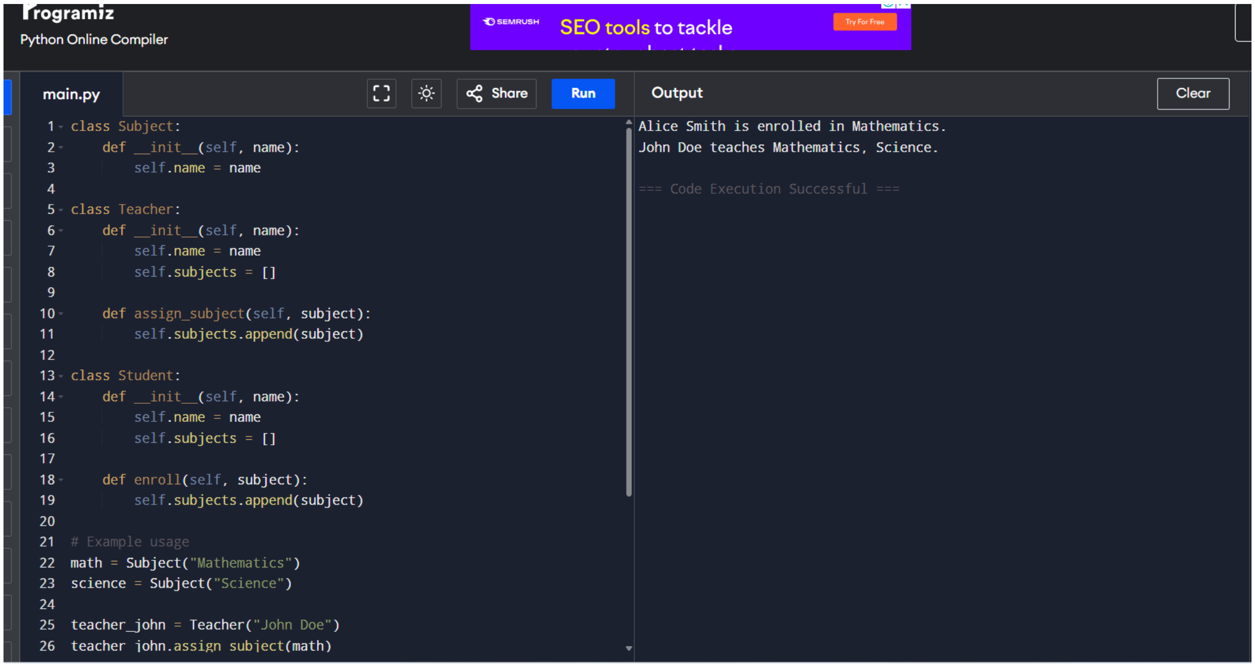
```
import sqlite3

# Connect to the database (or create it if it doesn't exist)
conn = sqlite3.connect('people.db')

# Create a cursor object
cursor = conn.cursor()

# Create a table for storing names and DOB
cursor.execute('''
CREATE TABLE IF NOT EXISTS person (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    dob DATE NOT NULL
)
''')

# Function to insert a new person into the database
def insert_person(name, dob):
    cursor.execute('''
    INSERT INTO person (name, dob) VALUES (?, ?)
    ''', (name, dob))
    conn.commit()

# Example usage
insert_person('John Doe', '1990-01-01')
```

```
(1, 'John Doe', '1990-01-01')
(2, 'Jane Smith', '1985-05-15')


=== Code Exited With Errors ===
```

19. Write the prolog program for STUDENT -TEACHER-SUB-CODE.

```
class Subject:
    def __init__(self, name):
        self.name = name

class Teacher:
    def __init__(self, name):
        self.name = name
        self.subjects = []

    def assign_subject(self, subject):
        self.subjects.append(subject)

class Student:
    def __init__(self, name):
        self.name = name
        self.subjects = []

    def enroll(self, subject):
        self.subjects.append(subject)

# Example usage
math = Subject("Mathematics")
science = Subject("Science")

teacher_john = Teacher("John Doe")
```

```
teacher_john.assign_subject(math)

teacher_john.assign_subject(science)


student_alice = Student("Alice Smith")

student_alice.enroll(math)


print(f"{student_alice.name} is enrolled in {', '.join([sub.name for sub in student_alice.subjects])}.")

print(f"{teacher_john.name} teaches {', '.join([sub.name for sub in teacher_john.subjects])}.")
```



```python
class Subject:
    def __init__(self, name):
        self.name = name

class Teacher:
    def __init__(self, name):
        self.name = name
        self.subjects = []

    def assign_subject(self, subject):
        self.subjects.append(subject)

class Student:
    def __init__(self, name):
        self.name = name
        self.subjects = []

    def enroll(self, subject):
        self.subjects.append(subject)

# Example usage
math = Subject("Mathematics")
science = Subject("Science")

teacher_john = Teacher("John Doe")
teacher_john.assign_subject(math)
```

```
Alice Smith is enrolled in Mathematics.
John Doe teaches Mathematics, Science.

=== Code Execution Successful ===
```

20. Write the prolog program for PLANETS DB.

```prolog
% Facts about planets

planet(mercury, terrestrial, 57.91e6, no).

planet(venus, terrestrial, 108.2e6, no).

planet(earth, terrestrial, 149.6e6, [moon]).

planet(mars, terrestrial, 227.9e6, [phobos, deimos]).

planet(jupiter, gas_giant, 778.5e6, [io, europa, ganymede, callisto]).
```

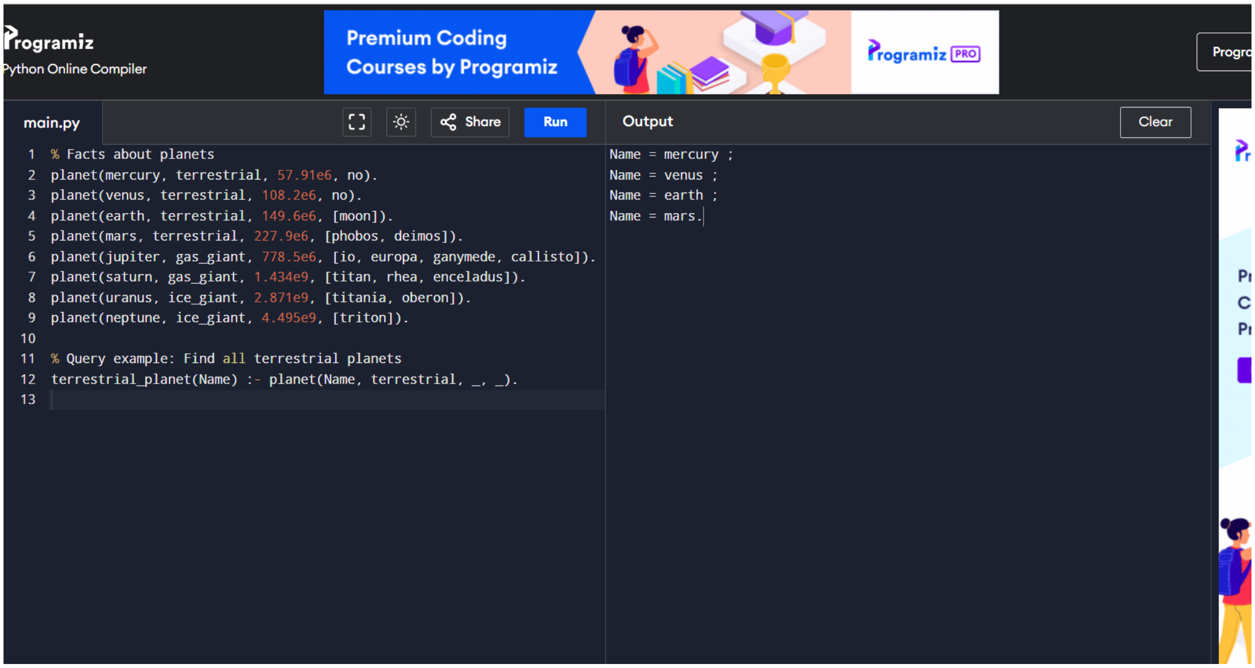
planet(saturn, gas_giant, 1.434e9, [titan, rhea, enceladus]).

planet(uranus, ice_giant, 2.871e9, [titania, oberon]).

planet(neptune, ice_giant, 4.495e9, [triton]).

% Query example: Find all terrestrial planets

terrestrial_planet(Name) :- planet(Name, terrestrial, _, _).



```
% Facts about planets
planet(mercury, terrestrial, 57.91e6, no).
planet(venus, terrestrial, 108.2e6, no).
planet(earth, terrestrial, 149.6e6, [moon]).
planet(mars, terrestrial, 227.9e6, [phobos, deimos]).
planet(jupiter, gas_giant, 778.5e6, [io, europa, ganymede, callisto]).
planet(saturn, gas_giant, 1.434e9, [titan, rhea, enceladus]).
planet(uranus, ice_giant, 2.871e9, [titania, oberon]).
planet(neptune, ice_giant, 4.495e9, [triton]).

% Query example: Find all terrestrial planets
terrestrial_planet(Name) :- planet(Name, terrestrial, _, _).
```

```
Name = mercury ;
Name = venus ;
Name = earth ;
Name = mars.
```